Encoding dependent types in an intuitionistic logic

Amy Felty

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.: (1) 39 63 55 11

Rapports de Recherche

N° 1521

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

ENCODING DEPENDENT TYPES IN AN INTUITIONISTIC LOGIC

Amy FELTY

Septembre 1991

RR-1521

Encoding Dependent Types in an Intuitionistic Logic

Amy Felty
INRIA Rocquencourt

Abstract Various languages have been proposed as specification languages for representing a wide variety of logics. The development of typed λ-calculi has been one approach toward this goal. The *logical framework* (LF), a λ-calculus with dependent types is one example of such a language. A small subset of intuitionistic logic with quantification over the simply typed λ-calculus has also been proposed as a framework for specifying general logics. The logic of *hereditary Harrop* formulas with quantification at all non-predicate types, denoted here as hh^ω, is such a meta-logic. In this paper, we show how to translate specifications in LF into hh^ω specifications in a direct and natural way, so that correct typing in LF corresponds to intuitionistic provability in hh^ω. In addition, we demonstrate a direct correspondence between proofs in these two systems. The logic of hh^ω can be implemented using such logic programming techniques as providing operational interpretations to the connectives and implementing unification on λ-terms. As a result, relating these two languages makes it possible to provide direct implementations of proof checkers and theorem provers for logics specified in LF.

Codage des Types Dépendants dans une Logique Intuitionniste

Résumé De nombreux langages ont été proposés comme langages de spécification de logiques, en particulier plusieurs λ-calculs typés (par exemple le système LF, *logical framework*) et des sous-ensembles de la logique intuitionniste avec quantification sur les types simples (par exemple notre système hh^ω). Le système hh^ω est la logique de formules *héréditaires de Harrop* avec quantification sur tous les types sauf les prédicats. Dans cet article, nous montrons que les spécifications en LF peuvent être traduites dans hh^ω d'une manière directe et naturelle, telle que les termes typables en LF correspondent exactement aux formules démontrables dans la logique intuitionniste de hh^ω. Nous montrons aussi qu'il y a une correspondance directe entre les preuves des deux systèmes. La logique de hh^ω peut être implémentée en utilisant des techniques de programmation logique, en particulier l'interprétation opérationnelle des connecteurs et l'unification entre λ-termes. Cela permet l'implémentation directe de la vérification des preuves et de la recherche de démonstrations des theorèmes dans les logiques specifiées en LF.

Encoding Dependent Types in an Intuitionistic Logic *

Amy Felty
INRIA Rocquencourt
Domaine de Voluceau
78153 Le Chesnay Cedex, France

Abstract

Various languages have been proposed as specification languages for representing a wide variety of logics. The development of typed λ-calculi has been one approach toward this goal. The *logical framework* (LF), a λ-calculus with dependent types is one example of such a language. A small subset of intuitionistic logic with quantification over the simply typed λ-calculus has also been proposed as a framework for specifying general logics. The logic of *hereditary Harrop* formulas with quantification at all non-predicate types, denoted here as hh^ω, is such a meta-logic. In this paper, we show how to translate specifications in LF into hh^ω specifications in a direct and natural way, so that correct typing in LF corresponds to intuitionistic provability in hh^ω. In addition, we demonstrate a direct correspondence between proofs in these two systems. The logic of hh^ω can be implemented using such logic programming techniques as providing operational interpretations to the connectives and implementing unification on λ-terms. As a result, relating these two languages makes it possible to provide direct implementations of proof checkers and theorem provers for logics specified in LF.

1 Introduction

The design of languages that can express a wide variety of logics has been the focus of much recent work. Such languages attempt to provide a general theory of inference systems that captures uniformities across different logics, so that they can be exploited in implementing theorem provers and proof systems. One approach to the design of such languages is the development of various typed λ-calculi. Examples that have been proposed include the AUTOMATH languages [4], type theories developed by Martin-Löf [16], the Logical Framework (LF) [10], and the Calculus of Constructions [3]. A second approach is the use of a simple intuitionistic logic as a meta-language for expressing a wide variety of logics. The Isabelle theorem prover [20] and the λProlog logic programming language [18] provide implementations of a common subset of intuitionistic logic, called hh^ω here, that can be used for this purpose.

*To appear in *Logical Frameworks*, Gérard Huet and Gordon Plotkin, eds., Cambridge University Press.

In this paper, we will illustrate a strong correspondence between one language in the first category and a language in the second. In particular, we shall show how the Logical Framework (LF), a typed λ-calculus with dependent types, has essentially the same expressive power as hh^ω. We do so by showing how to translate LF typing judgments into hh^ω formulas such that correct typing in LF corresponds to intuitionistic provability in hh^ω.

Both Isabelle and λProlog can turn specifications of logics into proof checkers and theorem provers by making use of the unification of simply typed λ-terms and goal-directed, tactic-style search. Thus, besides answering the theoretical question about the precise relationship between these two meta-languages, this translation also describes how LF specifications of object logics can be implemented within such systems.

The translation we present here extends a translation given in [9]. As in that paper, we consider a form of LF such that all terms in derivable assertions are in *canonical* form, a notion which corresponds to $\beta\eta$-long normal form in the simply typed λ-calculus. In the translation given there, the form of proofs was also greatly limited. As we will illustrate, although we also restrict the form of terms here, we retain essentially the same power of provability as in LF as presented in [10]. As a result, theorem provers implemented from the hh^ω specifications obtained from this translation have a greater degree of flexibility.

In the next section, we provide some further motivation for establishing a formal relation between these two meta-languages. Then, in Section 3 we present LF, and in Section 4 we present the meta-logic hh^ω. Section 5 presents a translation of LF into hh^ω and Section 6 contains a proof of its correctness. Section 7 provides examples of this translation using an LF specification of natural deduction for first-order logic. Finally, Section 8 concludes.

2 Motivation

Our objectives in defining an encoding from LF into hh^ω are both theoretical and practical. On the theoretical side, we hope that by providing an alternate presentation of LF (via its encoding into a different formalism), we can provide some insight into the information contained in dependent types. In addition, we wish to formally establish the correspondence between two different approaches to specifying general logics. On the practical side, as already mentioned, we wish to provide an approach to implementing proof checkers and theorem provers for logics specified in these meta-languages. We address both of these concerns below.

2.1 Dependent Types as Formulas

A dependent type in LF has the structure $\Pi x : A.B$ where A and B are types and x is a variable of type A bound in this expression. The type B may contain occurrences of x. This structure represents a "functional type." If f is a function of this type, and N is a term of type A, then fN (f applied to N) has the type B where all occurrences of x are replaced by N, often written $[N/x]B$. Thus the argument type is A and the result type *depends* on the value input to the function. Another way to read such a type is as follows: "for any element x, if x has type A then fx has type B." This reading suggests a

logical interpretation of such a type: "for any" suggests universal quantification while "if then" suggests implication. It is exactly this kind of "propositional content" of dependent types that will be made explicit by our encoding. When x does not occur in B, such a dependent type corresponds to the simple functional type $A \rightarrow B$. Note that the logical reading remains the same in this simpler case. For the case when x occurs in B, we can think of B as a predicate over x.

In the LF encoding of natural deduction for first-order logic, for example, first-order formulas are represented as LF terms of type *form* and a function *true* of type *form* $\rightarrow$ Type is defined which takes formulas into LF types. The constant *true* is used to encode the provability judgment of first-order logic: the type (*true* A) represents the statement "formula A is provable," and LF terms of this type are identified with natural deduction proofs for this formula. (This is an example of the LF "judgments as types" principle, similar to the "formulas as types" principle as in [14].) Via the encoding in hh^ω, we will view *true* as a predicate over first-order formulas. Proofs of the predicate (*true* A) in hh^ω can be identified with natural deduction proofs of A. Our results establish a very close connection between hh^ω proofs of such predicates and LF proofs of their corresponding typing judgments.

2.2 Implementing Goal Directed Search in Dependent-Type Calculi

In general, the search for terms inhabiting types in LF corresponds to object-level theorem proving. For example, searching for a term of type (*true* $C \wedge D$) corresponds to searching for a natural deduction proof of the conjunction $C \wedge D$. To find a term of this type we may use, for example, the following item which encodes the $\wedge$-introduction rule for natural deduction.

$$\wedge\text{-I} : \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.(\mathit{true}\ A) \rightarrow (\mathit{true}\ B) \rightarrow (\mathit{true}\ A \wedge B)$$

(We will say a type is "atomic" if it has no leading Π. We call the rightmost atomic type, (*true* $A \wedge B$) in this case, the "target" type. The types *form*, (*true* A), and (*true* B) are said to be "argument" types.) This type can be read: for any formulas A and B, if A is provable and B is provable, then the conjunction $A \wedge B$ is provable. Thus, if C and D indeed have type *form*, and there exist terms P and Q inhabiting types (*true* C) and (*true* D), then ($\wedge$-I C D P Q) is a term of the desired type.

Consider the following general goal directed approach to the search for an inhabiting term of a given type. If the type is atomic, attempt to match it with the target type of an existing axiom or hypothesis. If there is a match, attempt to find inhabitants of each of the argument types. If the type is of the form $\Pi x{:}A.B$, add $x{:}A$ as a new hypothesis, and attempt to find an inhabitant of B. It is exactly this kind of approach to search that we obtain via the translation. More specifically, our encoding will map each LF axiom such as the one above specifying the $\wedge$-introduction rule to an hh^ω formula. With respect to a logic programming interpreter implementing hh^ω that will be described in Section 4, search using such translated LF axioms will correspond exactly to the above description of goal directed search in LF.

We will see that a set of hh^ω formulas obtained by translating an LF representation of an object logic can serve directly as a proof checker for that logic. In other words, a

given hh^ω formula will be provable with respect to the depth-first interpreter implementing hh^ω described in Section 4 if and only if the corresponding LF typing judgment is provable in the LF type system. For theorem proving, or searching for a term inhabiting a type, more sophisticated control is necessary. In [7], it is shown that a theorem proving environment with tactic style search can be implemented in λProlog. The clauses obtained by the translation can serve as the basic operations to such a theorem prover. In fact, tactic theorem provers for many of the example LF specifications given in [1] have been implemented and tested in λProlog. Within such a tactic environment, more complex search strategies can be written from the basic operations. For example, for a theorem prover obtained by translating an LF specification of natural deduction, a simple tactic can be written that automates the application of introduction rules, performing all possible applications of such rules to a given input formula.

The hh^ω specification of natural deduction described in Section 7 obtained via translation is in fact quite similar to the direct specification given in [7]. Thus, the tactic theorem prover described in that paper is very similar in behavior to the one obtained via translation. One difference is that an alternate specification of the elimination rules is given in [7] such that goal-directed search in hh^ω corresponds to forward reasoning in natural deduction. Using this approach, it is possible to apply rules to existing hypotheses in a forward direction, a capability which is quite useful for theorem proving in natural deduction style systems. (See also [6] for more on this kind of reasoning in natural deduction and its correspondence to backward reasoning on the left in sequent style inference systems.) It is in fact straightforward to define an LF specification of natural deduction in first-order logic whose translation has the property that rules can be applied to hypotheses in a forward direction. Thus a goal directed strategy at the meta-level (LF or hh^ω) does not necessarily impose a goal directed strategy at the object-level.

3 The Logical Framework

There are three levels of terms in the LF type theory: objects (often called just terms), types and families of types, and kinds. We assume two given denumerable sets of variables, one for object-level variables and the other for type family-level variables. The syntax of LF is given by the following classes of objects.

$$\begin{array}{rcl} K & := & \mathsf{Type} \mid \Pi x{:}A.K \\ A & := & x \mid \Pi x{:}A.B \mid \lambda x{:}A.B \mid AM \\ M & := & x \mid \lambda x{:}A.M \mid MN \\ \Gamma & := & \langle\rangle \mid \Gamma, x{:}K \mid \Gamma, x{:}A \mid \Gamma, A{:}K \mid \Gamma, M{:}A \end{array}$$

Here M and N range over expressions for objects, A and B over types and families of types, K over kinds, x over variables, and Γ over contexts. The empty context is denoted by $\langle\rangle$. We will use P and Q to range over arbitrary objects, types, type families, or kinds. We write $A \rightarrow P$ for $\Pi x{:}A.P$ when x does not occur in type or kind P. We will say that a type or type family of the form $xN_1 \ldots N_n$ where $n \geq 0$ and x is a type family-level variable is a *flat type*.

Terms that differ only in the names of variables bound by λ or Π are identified. If x is an object-level variable and N is an object then $[N/x]$ denotes the operation of substituting N for all free occurrences of x, systematically changing bound variables in order to avoid variable capture. The expression $[N_1/x_1, \ldots, N_n/x_n]$ will denote the simultaneous substitution of the terms $N_1, \ldots, N_n$ for distinct variables $x_1, \ldots, x_n$, respectively.

The notion of β-conversion at the level of objects, types, type families, and kinds can be defined in the obvious way using the usual rule for β-reduction at the level of both objects and type families: $(\lambda x : A.P)N \rightarrow_\beta [N/x]P$ where P is either an object or type/type family. The relation of convertibility up to β is written as $=_\beta$. All well-typed LF terms are strongly normalizing [10]. We write P^β to denote the normal form of term P.

Let Q be a type or kind whose normal form is $\Pi x_1 : A_1 \ldots \Pi x_n : A_n.P$ where P is Type, a variable, or an application. We define the *order* of Q to be 0 if n is 0, and 1 greater than the maximum order of $A_1, \ldots, A_n$ otherwise.

We present a version of the LF proof system that constructs only terms in canonical form. Several definitions from [11] are required to establish this notion. We define the *arity* of a type or kind to be the number of Πs in the prefix of its normal form. The arity of a variable with respect to a context is the arity of its type in that context. The arity of a bound variable occurrence in a term is the arity of the type label attached to its binding occurrence. An occurrence of a variable x in a term is *fully applied* with respect to a context if it occurs in a subterm of the form $xM_1 \ldots M_n$, where n is the arity of x. A term P is *canonical* with respect to a context Γ if P is in β-normal form and every variable occurrence in P is fully applied with respect to Γ. A term P is *pre-canonical* if its β-normal form is canonical. Flat types $xN_1 \ldots N_n$ such that x is fully applied will be called *base types.*

The following four kinds of *assertions* are derivable in the LF type theory.

$$\begin{array}{ll} \vdash \Gamma \text{ context} & (\Gamma \text{ is valid context}) \\ \Gamma \vdash K \text{ kind} & (K \text{ is a kind in } \Gamma) \\ \Gamma \vdash A : K & (A \text{ has kind } K \text{ in } \Gamma) \\ \Gamma \vdash M : A & (M \text{ has type } A \text{ in } \Gamma) \end{array}$$

For the special form $\Gamma \vdash A :$ Type of the third type of assertion, we also say A is a type in Γ. For the latter three assertions, we say that K, A, or M, respectively, is a *well-typed term* in Γ. We write $\Gamma \vdash \alpha$ for an arbitrary assertion of one of these three forms, where α is called an LF *judgment.* In deriving an assertion of this form, we always assume that we start with a valid context Γ.

We extend the notation for substitution and β-normalization to contexts and judgments. We write $[N/x]\Gamma$ and $[N/x]\alpha$ to denote the substitution of N for x in all terms in context Γ and judgment α, respectively. Similarly, we write Γ^β and α^β to denote the context and judgment obtained by replacing every term in Γ and α, respectively, by their normal forms.

The inference rules of LF are given in Figures 1 and 2. The set of variables on the left of the colon in a context Γ is denoted as $\mathrm{dom}(\Gamma)$. In (FAM-INTRO), (OBJ-INTRO), (PI-KIND), (PI-FAM), (ABS-FAM), and (ABS-OBJ) in Figure 1, we assume that the variable x does not occur in Γ, and in (APP-FAM) and (APP-OBJ) in Figure 2, we assume that

$$\vdash \langle\rangle \text{ context} \quad \text{(EMPTY-CTX)}$$

$$\frac{\vdash \Gamma \text{ context} \qquad \Gamma \vdash K \text{ kind}}{\vdash \Gamma, x{:}K \text{ context}} \text{(FAM-INTRO)}$$

$$\frac{\vdash \Gamma \text{ context} \qquad \Gamma \vdash A : \mathsf{Type}}{\vdash \Gamma, x{:}A \text{ context}} \text{(OBJ-INTRO)}$$

$$\frac{\vdash \Gamma \text{ context} \qquad \Gamma \vdash A : K}{\vdash \Gamma, A{:}K \text{ context}} \text{(FAM-LEMMA)}$$

$$\frac{\vdash \Gamma \text{ context} \qquad \Gamma \vdash M : A}{\vdash \Gamma, M{:}A \text{ context}} \text{(OBJ-LEMMA)}$$

$$\Gamma \vdash \mathsf{Type} \text{ kind} \quad \text{(TYPE-KIND)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash K \text{ kind}}{\Gamma \vdash \Pi x{:}A.K \text{ kind}} \text{(PI-KIND)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B : \mathsf{Type}}{\Gamma \vdash \Pi x{:}A.B : \mathsf{Type}} \text{(PI-FAM)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B : K}{\Gamma \vdash \lambda x{:}A.B : \Pi x{:}A.K} \text{(ABS-FAM)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \qquad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B} \text{(ABS-OBJ)}$$

Figure 1: LF contexts and abstraction rules

the variables $x_1, \ldots, x_n$ do not occur free in $N_1, \ldots, N_n$. Note that bound variables can always be renamed to meet these restrictions. In addition, in (APP-OBJ) B must be a base type. Note that when B is a base type, so is $([N_1/x_1, \ldots, N_n/x_n]B)^\beta$.

Items introduced into contexts by (FAM-LEMMA) or (OBJ-LEMMA) will be called *context lemmas*. The main differences between this presentation and the usual presentation of the LF type system are the appearance of such lemmas in contexts and the form of the (APP-FAM) and (APP-OBJ) rules. Here, in any derivation, all terms that are used on the left of an application must occur explicitly in the context.

We say that a context Γ is canonical (pre-canonical) if for every item $x{:}P$ in Γ where x is a variable, P is canonical (pre-canonical), and for every context lemma $P{:}Q$ in Γ, both P and Q are canonical (pre-canonical) with respect to Γ. We say that an assertion is canonical (pre-canonical) if the context is canonical (pre-canonical) and all terms in the judgment on the left of the turnstile are canonical (pre-canonical). In this presentation, all derivable assertions are canonical. To see why, first note that no new β-redexes are introduced in the conclusion of any rule. Second, consider the application rules. In the

$$\frac{\begin{array}{l} B : \Pi x_1 : A_1 \ldots \Pi x_n : A_n.\mathsf{Type} \in \Gamma \\ \Gamma \vdash N_1 : A_1 \\ \Gamma \vdash N_2 : ([N_1/x_1]A_2)^\beta \\ \vdots \\ \Gamma \vdash N_n : ([N_1/x_1, \ldots, N_{n-1}/x_{n-1}]A_n)^\beta \end{array}}{\Gamma \vdash (BN_1 \ldots N_n)^\beta : \mathsf{Type}} \text{(APP-FAM)}$$

$$\frac{\begin{array}{l} M : \Pi x_1 : A_1 \ldots \Pi x_n : A_n.B \in \Gamma \\ \Gamma \vdash N_1 : A_1 \\ \Gamma \vdash N_2 : ([N_1/x_1]A_2)^\beta \\ \vdots \\ \Gamma \vdash N_n : ([N_1/x_1, \ldots, N_{n-1}/x_{n-1}]A_n)^\beta \end{array}}{\Gamma \vdash (MN_1 \ldots N_n)^\beta : ([N_1/x_1, \ldots, N_n/x_n]B)^\beta} \text{(APP-OBJ)}$$

Figure 2: LF application rules

(APP-OBJ) or (APP-FAM) rule, if the term on the left of the application is a variable x, then it has arity n and is applied in the conclusion to n terms and thus this occurrence of x is fully applied. Hence, as long as $N_1, \ldots, N_n$ are canonical, so is $xN_1 \ldots N_n$. If the term on the left of the application is a canonical term, then it has the form $\lambda x_1 : A_1 \ldots \lambda x_n : A_n.P$. The term in the conclusion has the form $((\lambda x_1 : A_1 \ldots \lambda x_n : A_n.P)N_1 \ldots N_n)^\beta$ which is equivalent to $([N_1/x_1, \ldots, N_n/x_n]P)^\beta$. The fact that this latter term is canonical follows from the fact that for any object, type, type family, or kind Q and any object N, if Q and N are canonical, then so is $([N/x]Q)^\beta$. For the same reason, the type $([N_1/x_1, \ldots, N_n/x_n]B)^\beta$ in the (APP-OBJ) rule is canonical.

In Appendix A, we show formally the correspondence between LF as presented in [10], which we call *full* LF, and LF as presented here, which we will call *canonical* LF. In full LF, terms in derivable judgments are not necessarily canonical or β-normal. For a provable assertion $\Gamma \vdash \alpha$ in full LF, we say that $\Gamma^\beta \vdash \alpha^\beta$ is its *normal form*. In Appendix A, we demonstrate that any derivation of a pre-canonical assertion in full LF can be mapped directly to a derivation in canonical LF of its normal form. Conversely, any derivation of $\Gamma \vdash \alpha$ in canonical LF has a corresponding derivation of a pre-canonical assertion in full LF whose normal form is $\Gamma \vdash \alpha$. It is important to emphasize that these results demonstrate not only a correspondence between what is provable in each system, but also a direct correspondence between derivations in each system. In other words, full LF restricted to pre-canonical terms is essentially the same system as canonical LF presented here.

It can now be seen how the goal directed strategy discussed in Section 2 can be applied to construct a proof of an LF assertion in this system. For example to find an object inhabiting an LF type, the (ABS-OBJ) rule is applied if the type has a leading Π, and the (APP-OBJ) rule is attempted if the type is atomic. In this case, goal directed proof corresponds to searching for a term in the context whose target type matches with the atomic type.

4 The Intuitionistic Logic hh^ω

The terms of the logic hh^ω are the simply typed λ-terms. Let S be a fixed, finite set of *primitive types*. We assume that the symbol o is always a member of S. Following Church [2], o is the type for propositions. The set of *types* is the smallest set of expressions that contains the primitive types and is closed under the construction of function types, denoted by the binary, infix symbol $\rightarrow$. The Greek letter τ is used as a syntactic variable ranging over types. The type constructor $\rightarrow$ associates to the right. If τ_0 is a primitive type then the type $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau_0$ has $\tau_1, \ldots, \tau_n$ as *argument types* and τ_0 as *target type*. The *order* of a primitive type is 0 while the order or a non-primitive type is one greater than the maximum order of its argument types.

For each type τ, we assume that there are denumerably many constants and variables of that type. Constants and variables do not overlap and if two constants (variables) have different types, they are different constants (variables). A *signature* is a finite set Σ of constants and variables whose types are such that their argument types do not contain o. A constant with target type o is a *predicate constant*.

Simply typed λ-terms are built in the usual way. An abstraction is written as $\lambda x\ t$, or $\lambda x : \tau . t$ when we wish to be explicit about the type of the bound variable x. The logical constants are given the following types: $\wedge$ (conjunction) and $\supset$ (implication) are both of type $o \rightarrow o \rightarrow o$; $\top$ (true) is of type o; and $\forall_\tau$ (universal quantification) is of type $(\tau \rightarrow o) \rightarrow o$, for all types τ not containing o. A formula is a term of type o. The logical constants $\wedge$ and $\supset$ are written in the familiar infix form. The expression $\forall_\tau(\lambda x\ t)$ is written $\forall_\tau x\ t$ or simply as $\forall x\ t$ when types can be inferred from context.

If x and t are terms of the same type then $[t/x]$ denotes the operation of substituting t for all free occurrences of x, systematically changing bound variables in order to avoid variable capture. The expression $[t_1/x_1, \ldots, t_n/x_n]$ will denote the simultaneous substitution of the terms $t_1, \ldots, t_n$ for distinct variables $x_1, \ldots, x_n$, respectively.

We shall assume that the reader is familiar with the usual notions and properties of α, β, and η conversion for the simply typed λ-calculus. The relation of convertibility up to α and β is written as $=_\beta$ (as it is for LF), and if η is added, is written as $=_{\beta\eta}$. A λ-term is in β-normal form if it contains no beta redexes, that is, subformulas of the form $(\lambda x\ t)s$. We say that an occurrence of a variable or constant h in a simply typed λ-term is *fully applied* if it occurs in a subterm of the form $ht_1 \ldots t_n$ having primitive type. The term h is called the *head* of this subterm. A λ-term is in $\beta\eta$-long form if it is in β-normal form and every variable and constant occurrence is fully applied. All λ-terms $\beta\eta$-convert to a term in $\beta\eta$-long form, unique up to α-conversion. See [13] for a fuller discussion of these basic properties of the simply typed λ-calculus.

Let Σ be a signature. A term is a Σ-*term* if all of its free variables and nonlogical constants are members of Σ. Similarly, a formula is a Σ-*formula* if all of its free variables and nonlogical constants are members of Σ. A formula is either *atomic* or *non-atomic*. An atomic Σ-formula is of the form $(Pt_1 \ldots t_n)$, where $n \geq 0$, P is given type $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow o$ by Σ, and $t_1, \ldots, t_n$ are terms of the types $\tau_1, \ldots, \tau_n$, respectively. The predicate constant P is the *head* of this atomic formula. Non-atomic formulas are of the form $\top$, $B_1 \wedge B_2$, $B_1 \supset B_2$, or $\forall_\tau x\ B$, where B, B_1, and B_2 are formulas.

The logic we have just presented is very closely related to two logic programming

$$\frac{\Sigma\ ;\ B, C, \mathcal{P} \longrightarrow C}{\Sigma\ ;\ B \wedge C, \mathcal{P} \longrightarrow C}\ \wedge\text{-L} \qquad \frac{\Sigma\ ;\ \mathcal{P} \longrightarrow B \qquad \Sigma\ ;\ \mathcal{P} \longrightarrow C}{\Sigma\ ;\ \mathcal{P} \longrightarrow B \wedge C}\ \wedge\text{-R}$$

$$\frac{\Sigma\ ;\ \mathcal{P} \longrightarrow B \qquad \Sigma\ ;\ C, \mathcal{P} \longrightarrow A}{\Sigma\ ;\ B \supset C, \mathcal{P} \longrightarrow A}\ \supset\text{-L} \qquad \frac{\Sigma\ ;\ B, \mathcal{P} \longrightarrow C}{\Sigma\ ;\ \mathcal{P} \longrightarrow B \supset C}\ \supset\text{-R}$$

$$\frac{\Sigma\ ;\ [t/x]B, \mathcal{P} \longrightarrow C}{\Sigma\ ;\ \forall_\tau x\ B, \mathcal{P} \longrightarrow C}\ \forall\text{-L} \qquad \frac{\Sigma \cup \{c\}\ ;\ \mathcal{P} \longrightarrow [c/x]B}{\Sigma\ ;\ \mathcal{P} \longrightarrow \forall_\tau x\ B}\ \forall\text{-R}$$

Figure 3: Left and right introduction rules for hh^ω

extensions that have been studied elsewhere [17]. *First-order hereditary Harrop* formulas (*fohh*) have been studied as an extension to first-order Horn clauses as a basis for logic programming. Similarly *higher-order hereditary Harrop* formulas (*hohh*) are a generalization of *fohh* that permits some forms of predicate quantification. Because our meta-language is neither higher-order, since it lacks predicate quantification, nor first-order, since it contains quantification at all function types, we shall simply call it hh^ω. The set of hh^ω formulas in which quantification only up to order n is used will be labeled as hh^n.

Provability for hh^ω can be given in terms of sequent calculus proofs. A *sequent* is a triple $\Sigma\ ;\ \mathcal{P} \longrightarrow B$, where Σ is a signature, B is a Σ-formula, and $\mathcal{P}$ is a finite (possibly empty) sets of Σ-formulas. The set $\mathcal{P}$ is this sequent's *antecedent* and B is its *succedent*. Later, when discussing an interpreter for this language, we also say that $\mathcal{P}$ is a *program*, that each formula in $\mathcal{P}$ is a *clause*, and that B is a *goal formula*. The expression $B, \mathcal{P}$ denotes the set $\mathcal{P} \cup \{B\}$; this notation is used even if $B \in \mathcal{P}$. The inference rules for sequents are presented in Figure 3. The following provisos are also attached to the two inference rules for quantifier introduction: in $\forall$-R c is a constant of type τ not in Σ, and in $\forall$-L t is a Σ-term of type τ.

A *proof* of the sequent $\Sigma\ ;\ \mathcal{P} \longrightarrow B$ is a finite tree constructed using these inference rules such that the root is labeled with $\Sigma\ ;\ \mathcal{P} \longrightarrow B$ and the leaves are labeled with *initial sequents*, that is, sequents $\Sigma'\ ;\ \mathcal{P}' \longrightarrow B'$ such that either B' is $\top$ or $B' \in \mathcal{P}'$. The non-terminals in such a tree are instances of the inference figures in Figure 3. Since we do not have an inference figure for $\beta\eta$-conversion, we shall assume that in building a proof, two formulas are equal if they are $\beta\eta$-convertible. If the sequent $\Sigma\ ;\ \mathcal{P} \longrightarrow B$ has a sequent proof then we write $\Sigma; \mathcal{P} \vdash_I B$ and say that B is provable from Σ and $\mathcal{P}$. The following two theorems establish the main proof theoretic results of hh^ω we shall need. These theorems are direct consequences of the proof theory of a more expressive logic studied in [17].

Theorem 1 Let Σ be a signature, let $\mathcal{P}$ be a finite set of Σ-formulas, and let B be a Σ-formula. The sequent $\Sigma\ ;\ \mathcal{P} \longrightarrow B$ has a proof if and only if it has a proof in which

every sequent containing a non-atomic formula as its succedent is the conclusion of a right introduction rule.

To state our second theorem, we need the following definition.

Definition 2 Let Σ be a signature and let $\mathcal{P}$ be a finite set of Σ-formulas. The expression $|\mathcal{P}|_\Sigma$ denotes the smallest set of pairs $\langle \mathcal{G}, D\rangle$ of finite set of Σ-formulas $\mathcal{G}$ and Σ-formula D, such that

- If $D \in \mathcal{P}$ then $\langle \emptyset, D\rangle \in |\mathcal{P}|_\Sigma$.
- If $\langle \mathcal{G}, D_1 \wedge D_2\rangle \in |\mathcal{P}|_\Sigma$ then $\langle \mathcal{G}, D_1\rangle \in |\mathcal{P}|_\Sigma$ and $\langle \mathcal{G}, D_2\rangle \in |\mathcal{P}|_\Sigma$.
- If $\langle \mathcal{G}, \forall_\tau x D\rangle \in |\mathcal{P}|_\Sigma$ then $\langle \mathcal{G}, [t/x]D\rangle \in |\mathcal{P}|_\Sigma$ for all Σ-terms t of type τ.
- If $\langle \mathcal{G}, G \supset D\rangle \in |\mathcal{P}|_\Sigma$ then $\langle \mathcal{G} \cup \{G\}, D\rangle \in |\mathcal{P}|_\Sigma$.

Theorem 3 Let Σ be a signature, let $\mathcal{P}$ be a finite set of Σ-formulas, and let A be an atomic Σ-formula. Then A is provable from Σ and $\mathcal{P}$ if and only if there is a pair $\langle \mathcal{G}, A\rangle \in |\mathcal{P}|_\Sigma$ so that for each $G \in \mathcal{G}$, G is provable from Σ and $\mathcal{P}$.

Given these two theorems, it is clear how a non-deterministic search procedure for hh^ω can be organized using the following four search primitives.

AND: $B_1 \wedge B_2$ is provable from Σ and $\mathcal{P}$ if and only if both B_1 and B_2 are provable from Σ and $\mathcal{P}$.

GENERIC: $\forall_\tau x B$ is provable from Σ and $\mathcal{P}$ if and only if $[c/x]B$ is provable from $\Sigma \cup \{c\}$ and $\mathcal{P}$ for any constant c of type τ not in Σ.

AUGMENT: $B_1 \supset B_2$ is provable from Σ and $\mathcal{P}$ if and only if B_2 is provable from Σ and $\mathcal{P} \cup \{B_1\}$.

BACKCHAIN: The atomic formula A is provable from Σ and $\mathcal{P}$ if and only if there is a pair $\langle \mathcal{G}, A\rangle \in |\mathcal{P}|_\Sigma$ so that for every $G \in \mathcal{G}$, G is provable from Σ and $\mathcal{P}$.

To implement an interpreter which implements these search operations, choices must be made which are left unspecified in the high-level description above. Here, we assume choices as in the λProlog language. For example, logic variables are employed in the BACKCHAIN operation to create universal instances of definite clauses. As a result, unification on λ-terms is necessary since logic variables of arbitrary functional type can occur inside λ-terms. Also the equality of terms is not a simple syntactic check but a more complex check of $\beta\eta$-conversion. Unification on λ-terms is not in general decidable. In λProlog, this issue is addressed by implementing a depth-first version of the unification search procedure described in [15]. (See [19, 17].) In this paper, the unification problems that result from programs we present are all decidable and rather simple.

In the AUGMENT search operation, clauses get added to the program dynamically. Note that as a result, clauses may in fact contain logic variables. The GENERIC operation must be implemented so that the new constant c introduced for x, must not appear in the terms eventually instantiated for logic variables free in the goal or in the program when c is introduced.

A deterministic interpreter must also specify the order in which conjuncts are attempted and definite clauses are backchained over. One possibility is to attempt conjuncts and backchain on definite clauses in the order in which they appear in the goal or in $\mathcal{P}$, respectively, using a depth-first search paradigm to handle failures as in Prolog.

5 Translating LF Assertions to hh^ω Formulas

In this section we present the translation of LF assertions to formulas in hh^ω. This translation will require an encoding of LF terms as simply typed λ-terms. We begin by presenting this encoding. We then present the translation, which has three parts. The first translates context items to a set of hh^ω formulas to be used as assumptions, while the second translates LF judgments to a formula to be proven with respect to such a set of assumptions. The third translation is defined using the previous two, and translates an LF assertion $\Gamma \vdash \alpha$ to a single formula whose proof verifies that Γ is a valid context before proving that α holds within the context Γ.

In this section, since we encode LF in hh^ω, we consider hh^ω as the meta-language and LF as the object-language. Since both languages have types and terms, to avoid confusion we will refer to types and terms of hh^ω as *meta-types* and *meta-terms*. In order to define an encoding of LF terms as simply typed λ-terms, we change slightly the notion of LF syntax. We will associate to each object and type variable, a tag which indicates the "syntactic structure" of types and kinds, respectively, which can be associated with it in forming a binder or a context item. These tags will be "simple types" built up from two primitive types `ob` and `ty` and the arrow constructor $\rightarrow$, with the additional restriction that `ty` can only appear as a target type. We assume that there is an infinite number of object-level and type-level variables associated with every simple type whose target type is `ob` and `ty`, respectively.

Let x be an object or type variable and $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau_0$ be the tag associated with x, where $n \geq 0$ and τ_0 is `ob` or `ty`. We say that variable x *admits* type or kind $\Pi x_1 : A_1 \ldots x_n : A_n.P$ if the following hold: if τ_0 is `ty` then P is Type; if τ_0 is `ob`, then P is a flat type; for $i = 1, \ldots, n$, the tag associated with x_i is τ_i and x_i admits type A_i. We add a restriction when forming the λ or Π binder $x : A$, or the context item $x : A$ or $x : K$, that x admits type A or kind K. Note that this restriction requires that the "simple type" in a variable tag has exactly the same order as the LF type or kind used in forming the binder.

We only define the encoding of LF terms as simply typed λ-terms for LF objects and flat types since this is all that is required by the translation. We introduce two primitive types at the meta-level, *ob* and *ty*, for these two classes of LF terms. The types `ob` and `ty` in variable tags correspond to these meta-types in the obvious way. When we wish to be explicit, we write $\mathcal{T}(x)$ to denote the meta-type associated with the tag on LF variable x obtained by replacing each occurrence of `ob` by *ob* and `ty` by *ty*. We will assume a fixed

mapping ρ which associates each LF variable x to a meta-variable of type $\mathcal{T}(x)$. For readability in our presentation, this mapping will be implicit. A variable x will represent both an LF variable and its corresponding meta-variable. It will always be clear from context which is meant.

We denote the encoding of term or type P as $\langle\!\langle P \rangle\!\rangle$. The full encoding is defined in Figure 4. Note that the encoding maps abstraction in LF objects directly to abstraction

$$\begin{array}{rcl} \langle\!\langle x \rangle\!\rangle & := & x \\ \langle\!\langle \lambda x : A.M \rangle\!\rangle & := & \lambda x : \mathcal{T}(x).\langle\!\langle M \rangle\!\rangle \\ \langle\!\langle MN \rangle\!\rangle & := & \langle\!\langle M \rangle\!\rangle \ \langle\!\langle N \rangle\!\rangle \\ \langle\!\langle AM \rangle\!\rangle & := & \langle\!\langle A \rangle\!\rangle \ \langle\!\langle M \rangle\!\rangle \end{array}$$

Figure 4: Encoding of LF Terms

at the meta-level, and that both application of objects to objects and application of type families to objects are mapped directly to application at the meta level. The difference at the meta-level is that the former application will be a meta-term with target type *ob* while the latter application will be a meta-term with target type *ty*.

We can easily define a function Φ which maps an LF type or kind to the simple type corresponding to the tag on a variable that admits this type or kind: $\Phi(\Pi x : A.P)$ is $\Phi(A) \rightarrow \Phi(P)$, $\Phi(\mathsf{Type})$ is *ty*, and $\Phi(xN_1 \ldots N_n)$ is *ob*. It is easy to see that for object or type family P having, respectively, type or kind Q, $\langle\!\langle P \rangle\!\rangle$ is a meta-term of meta-type $\Phi(Q)$.

Two predicates will appear in the atomic hh^ω formulas resulting from the translation: *hastype* of type $ob \rightarrow ty \rightarrow o$ and *istype* of type $ty \rightarrow o$. We will name the signature containing these two predicates Σ_{LF}. We denote the translation of the context item $P:Q$ as $[\![P:Q]\!]^+$. This translation is defined in Figure 5 (a). It is a partial function since it is defined by cases and undefined when no case applies. It will in fact always be defined on valid context items. When applied to a valid context item, P in the first two clauses in Figure 5 (a) will always be either an object or type family, and Q a type or kind, respectively. As was noted earlier, valid contexts are always in canonical form. Note that in a canonical context item $x:P$, the variable x is not necessarily canonical since it may not be fully applied. Such judgments with non-canonical terms on the left are handled by the second clause of the definition. Note the direct mapping of Π-abstraction in LF types and kinds to instances of universal quantification and implication in hh^ω formulas, as discussed earlier. In the first two clauses of the definition, the variable bound by Π is mapped to a variable at the meta-level bound by universal quantification. Then, in the resulting implication, the left hand side asserts the fact that the bound variable has a certain type, while the right hand side contains the translation of the body of the type or kind which may contain occurrences of this bound variable. The base cases occur when there is no leading Π in the type or kind, resulting in atomic formulas for the *hastype* and *istype* predicates.

To illustrate this translation, we consider an example from an LF context specifying natural deduction for first-order logic. The following context item introduces the constant for universal quantification and gives it a type: $\forall^* : (i \rightarrow form) \rightarrow form$. (We write $\forall^*$ for universal quantification at the object-level to distinguish it from universal

$$
\begin{aligned}
[\![\lambda x{:}A.P : \Pi x{:}A.Q]\!]^+ &:= \forall x \left([\![x{:}A]\!]^+ \supset [\![P : Q]\!]^+ \right) \\
[\![P : \Pi x{:}A.Q]\!]^+ &:= \forall x \left([\![x{:}A]\!]^+ \supset [\![Px : Q]\!]^+ \right) \\
&\quad \text{where } P \text{ is not an abstraction.} \\
[\![M : A]\!]^+ &:= \mathit{hastype}\ \langle\!\langle M \rangle\!\rangle\ \langle\!\langle A \rangle\!\rangle \\
&\quad \text{where } A \text{ is a flat type.} \\
[\![A : \mathsf{Type}]\!]^+ &:= \mathit{istype}\ \langle\!\langle A \rangle\!\rangle \quad \text{where } A \text{ is a flat type.}
\end{aligned}
$$

(a) Translation of context items

$$
\begin{aligned}
[\![\lambda x{:}A.P : \Pi x{:}A.Q]\!]^- &:= [\![A : \mathsf{Type}]\!]^- \wedge \forall x \left([\![x{:}A]\!]^+ \supset [\![P : Q]\!]^- \right) \\
[\![M : A]\!]^- &:= \mathit{hastype}\ \langle\!\langle M \rangle\!\rangle\ \langle\!\langle A \rangle\!\rangle \\
&\quad \text{where } A \text{ is a flat type.} \\
[\![A : \mathsf{Type}]\!]^- &:= \mathit{istype}\ \langle\!\langle A \rangle\!\rangle \quad \text{where } A \text{ is a flat type.} \\
[\![\Pi x{:}A.B : \mathsf{Type}]\!]^- &:= [\![A : \mathsf{Type}]\!]^- \wedge \forall x \left([\![x{:}A]\!]^+ \supset [\![B : \mathsf{Type}]\!]^- \right) \\
[\![\mathsf{Type\ kind}]\!]^- &:= \top \\
[\![\Pi x{:}A.K\ \mathrm{kind}]\!]^- &:= [\![A : \mathsf{Type}]\!]^- \wedge \forall x \left([\![x{:}A]\!]^+ \supset [\![K\ \mathrm{kind}]\!]^- \right)
\end{aligned}
$$

(b) Translation of LF judgments

Figure 5: Translating LF contexts and judgments to hh^ω formulas

quantification in hh^ω.) To make all bound variables explicit, we expand the above type to its unabbreviated form: $\Pi A : (\Pi y : i.\mathit{form}).\mathit{form}$. Note that the tag associated to $\forall^*$ must be $(\mathsf{ob} \to \mathsf{ob}) \to \mathsf{ob}$. Both the LF type and the corresponding meta-type have order 2. The translation of this context item is as follows.

$$
[\![\forall^* : \Pi A{:}(\Pi y{:}i.\mathit{form}).\mathit{form}]\!]^+ \equiv
$$

$$
\forall A \left(\forall y \left([\![y : i]\!]^+ \supset [\![Ay : \mathit{form}]\!]^+ \right) \supset [\![\forall^* A : \mathit{form}]\!]^+ \right) \equiv
$$

$$
\forall A\ (\forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ (Ay)\ \mathit{form})) \supset (\mathit{hastype}\ (\forall^* A)\ \mathit{form}))
$$

This formula provides the following description of the information contained in the above dependent type: for any A, if for arbitrary y of type i, Ay is a formula, then $\forall^* A$ is a formula.

Figure 5 (b) contains the definition of the translation for LF judgments. The translation of judgment α is denoted $[\![\alpha]\!]^-$. Several clauses of this definition are similar to the clauses of the previous one. For example, judgments containing a λ and Π-abstraction pair again translate to a universally quantified implication (using the first clause of the

$$
\begin{aligned}
[\![x:A,\Gamma;\alpha]\!] &:= [\![A:\mathsf{Type}]\!]^- \wedge \forall x \left([\![x:A]\!]^+ \supset [\![\Gamma;\alpha]\!]\right) \\
[\![x:K,\Gamma;\alpha]\!] &:= [\![K\ \mathsf{kind}]\!]^- \wedge \forall x \left([\![x:K]\!]^+ \supset [\![\Gamma;\alpha]\!]\right) \\
[\![P:Q,\Gamma;\alpha]\!] &:= [\![P:Q]\!]^- \wedge \left([\![P:Q]\!]^+ \supset [\![\Gamma;\alpha]\!]\right) \\
[\![\langle\rangle;\alpha]\!] &:= [\![\alpha]\!]^-
\end{aligned}
$$

Figure 6: Translating LF assertions to hh^ω formulas

definition). In this case, an additional conjunct is also required to verify that the type in the abstraction is valid. Note that in the left-hand side of the implication, the binder $x:A$ is translated as a context item (using $[\![]\!]^+$), and represents an additional assumption available when proving that P has type or kind Q. Since the term on the left of a colon in a canonical assertion is always canonical, we do not need a clause corresponding to the second clause of $[\![]\!]^+$. Note that the base cases resulting in atomic formulas for *hastype* and *istype* are identical to those for translating context items. Finally, the last three clauses handle the remaining possible LF judgments. Again Π-abstraction maps to universal quantification and implication, with an additional conjunct to verify that the type in the binder is valid. The judgment Type kind simply maps to $\top$.

Figure 6 contains the general translation for LF assertions. Given assertion $\Gamma \vdash \alpha$, the pair $(\Gamma;\alpha)$ is mapped to a single formula containing subformulas whose proofs will insure that each context item is valid and that the judgment holds in this context. The translation of such a pair is denoted $[\![\Gamma;\alpha]\!]$. The first two clauses of this translation map each context item to a conjunctive formula where the first conjunct verifies that the type or kind is valid (using the translation on LF judgments), and the second conjunct is a universally quantified implication where the left hand side asserts the fact that the context item has the corresponding type (using the translation on contexts), and the right side contains the translation of the pair consisting of the remaining context items and judgment. The third clause handles context lemmas. Again there are two conjuncts. The first translates the lemma as a judgment to verify that it holds, while the second translates it as a context item which will be available as an assumption in proving that the rest of the context is valid and that the judgment holds within the entire context. The last clause in the translation is for the base case. When the context is empty, the judgment is simply translated using $[\![]\!]^-$. In the next section, we will show formally that for LF assertion $\Gamma \vdash \alpha$, Γ is a valid context and $\Gamma \vdash \alpha$ is provable in LF if and only if $[\![\Gamma;\alpha]\!]$ is a provable hh^ω formula.

6 Correctness of Translation

The following two properties hold for the encoding $\langle\!\langle\rangle\!\rangle$ on terms. They will be important for establishing the correctness of the translation.

Lemma 4 Let P be an LF object or base type, and N an LF object. Then $[\langle\!\langle N\rangle\!\rangle/x]\langle\!\langle P\rangle\!\rangle = \langle\!\langle [N/x]P\rangle\!\rangle$.

Lemma 5 Let P and Q be two LF objects or base types. If $P =_\beta Q$, then $\langle\!\langle P \rangle\!\rangle =_\beta \langle\!\langle Q \rangle\!\rangle$.

Lemma 4 is proved by induction on the structure of LF terms, while Lemma 5 is proved by induction on a sequence of β-reductions to convert P to Q.

In proving the correctness of the translation, we consider a slightly modified LF. Our modified system replaces the (ABS-FAM) and (ABS-OBJ) rules with the following two rules.

$$\frac{\Gamma, x{:}A \vdash B : K}{\Gamma \vdash \lambda x{:}A.B : \Pi x{:}A.K}\ (\text{ABS-FAM}') \qquad \frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}\ (\text{ABS-OBJ}')$$

These rules are the same as presented earlier except that the left premise is omitted. We call this system LF$'$. Proving an assertion of the form $\Gamma \vdash \lambda x{:}A.P : \Pi x{:}A.Q$ in valid context Γ in the unmodified version of LF is equivalent to proving $\Gamma, x{:}A \vdash P : Q$ in valid context $\Gamma, x{:}A$. In Appendix B, we show that for valid context Γ and judgment α, the assertion $\Gamma \vdash \alpha$ is provable in LF if and only if it is provable in LF$'$ and all types bound by outermost abstractions in the term on the left in α are valid.

In proving correctness of the translation, we prove a stronger statement from which correctness will follow directly. This stronger statement will talk about the provability of an arbitrary LF$'$ assertion $\Gamma \vdash \alpha$ even in the case when Γ and the types bound by outermost abstractions in α are not valid. We make the following modifications to the definition of $[\![]\!]^-$ for translating such judgments: we replace the first clause of Figure 5 (b) with the first clause below, and add the second as a new clause.

$$\begin{aligned}
[\![\lambda x{:}A.P : \Pi x{:}A.Q]\!]^- &:= \forall x \left([\![x{:}A]\!]^+ \supset [\![P : Q]\!]^- \right) \\
[\![P : \Pi x{:}A.Q]\!]^- &:= \forall x \left([\![x{:}A]\!]^+ \supset [\![Px : Q]\!]^- \right) \\
&\qquad \text{where } P \text{ is not an abstraction.}
\end{aligned}$$

In the first clause, the removal of the left conjunct in these formulas corresponds to the removal of the left premise in the (ABS) rules. The second clause will be needed for proving our general form of the correctness theorem. Note that with these two clauses, the positive and negative translation are identical on judgments for which they are both defined.

One further lemma about LF$'$ is needed to prove the correctness of the (modified) translation. Lemma 4 shows that substitution commutes with the encoding operation. The lemma below extends this result to the translation operation on judgments which translate to provable hh^ω formulas. Given a context Γ, we write $\rho(\Gamma)$ to denote the set of meta-variables obtained by mapping, for each signature item $x : P$ in Γ, the variable x to the corresponding meta-variable of type $\mathcal{T}(x)$. We write $[\![\Gamma]\!]^+$ to denote the set of formulas obtained by translating separately each item in Γ using $[\![]\!]^+$.

Lemma 6 Let $\Gamma, x_1 : A_1, \ldots, x_n : A_n, x : A$ $(n \geq 0)$ be a canonical context. Let $N_1, \ldots, N_n, N$ be canonical objects with respect to Γ. Let Σ be the signature $\Sigma_{LF} \cup \rho(\Gamma)$. Then $\Sigma; [\![\Gamma]\!]^+ \vdash_I [\![N : ([N_1/x_1, \ldots, N_n/x_n]A)^\beta]\!]^-$ iff

$$\Sigma; [\![\Gamma]\!]^+ \vdash_I [\langle\!\langle N_1 \rangle\!\rangle / x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle / x_n, \langle\!\langle N \rangle\!\rangle / x][\![x : A]\!]^-.$$

Proof: The forward and backward direction is proved by simultaneous induction on the structure of A with the following statement. Let C be any $\Sigma \cup \{x\}$-formula. Then

$$\Sigma \cup \{x\}; [\![\Gamma]\!]^+, [\![x : ([N_1/x_1, \ldots, N_n/x_n]A)^\beta]\!]^+ \vdash_I C \quad \text{iff}$$

$$\Sigma \cup \{x\}; [\![\Gamma]\!]^+, [\langle\!\langle N_1 \rangle\!\rangle/x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle/x_n][\![x : A]\!]^+ \vdash_I C.$$

■

Theorem 7 (Correctness of Translation I) Let Γ be a valid context and α a canonical judgment such that all types bound by outermost abstractions in the term on the left in α are valid. Let Σ be $\Sigma_{LF} \cup \rho(\Gamma)$. Then $\Gamma \vdash \alpha$ is provable in LF′ iff $\Sigma; [\![\Gamma]\!]^+ \vdash_I [\![\alpha]\!]^-$ holds.

Proof: We prove a modified form of the above statement from which the theorem will follow directly. We relax the requirement that Γ is valid and that the types bound by outermost abstractions in α are valid. Instead, we simply require that $[\![\Gamma]\!]^+$ and $[\![\alpha]\!]^-$ are well-defined.

The proof of this theorem is constructive, *i.e.*, it provides a method for constructing an hh^ω proof from an LF proof, and vice versa. We begin with the forward direction which is proved by induction on the height of an LF′ proof of the assertion $\Gamma \vdash \alpha$. For the one node proof $\Gamma \vdash \mathsf{Type}$ kind, clearly $[\![\mathsf{Type}\ \mathrm{kind}]\!]^- \equiv \top$ is provable from Σ and $[\![\Gamma]\!]^+$. For the case when the last rule is (PI-FAM), we build the following sequent proof fragment, where the root is the translation of the conclusion of the (PI-FAM) rule, and the leaves are the translations of the premises which we know to be provable by the induction hypothesis.

$$\dfrac{\Sigma\,;\, [\![\Gamma]\!]^+ \longrightarrow [\![A : \mathsf{Type}]\!]^- \quad \dfrac{\dfrac{\Sigma \cup \{x\}\,;\, [\![\Gamma, x{:}A]\!]^+ \longrightarrow [\![B : \mathsf{Type}]\!]^-}{\Sigma \cup \{x\}\,;\, [\![\Gamma]\!]^+ \longrightarrow [\![x{:}A]\!]^+ \supset [\![B : \mathsf{Type}]\!]^-}\,\supset\text{-R}}{\Sigma\,;\, [\![\Gamma]\!]^+ \longrightarrow \forall x\,([\![x{:}A]\!]^+ \supset [\![B : \mathsf{Type}]\!]^-)}\,\forall\text{-R}}{\Sigma\,;\, [\![\Gamma]\!]^+ \longrightarrow [\![A : \mathsf{Type}]\!]^- \wedge \forall x\,([\![x{:}A]\!]^+ \supset [\![B : \mathsf{Type}]\!]^-)}\,\wedge\text{-R}$$

The case when the last rule is (PI-KIND) is similar. The cases for (ABS-OBJ′) and (ABS-FAM′) are also similar, except that the translations do not have the left conjunct and the corresponding LF′ proofs have only one premise. Next, consider the case when the last rule is (APP-OBJ) with context lemma $M : \Pi x_1 : A_1 \ldots \Pi x_n : A_n.B \in \Gamma$ and objects $N_1, \ldots, N_n$ appearing on the right of the colon in the n premises. We must show that the formula below (the translation of the conclusion) is provable from Σ and $[\![\Gamma]\!]^+$.

$$(\mathit{hastype}\ \langle\!\langle (M N_1 \ldots N_n)^\beta \rangle\!\rangle\ \langle\!\langle ([N_1/x_1, \ldots, N_n/x_n]B)^\beta \rangle\!\rangle) \tag{1}$$

(Note that we can assume that $x_1, \ldots, x_n$ do not appear free in M, otherwise we rename them in the above type.) By the induction hypothesis for the n premises and Lemma 6, the following are provable from Σ and $[\![\Gamma]\!]^+$.

$$[\langle\!\langle N_1 \rangle\!\rangle/x_1][\![x_1 : A_1]\!]^-, \ldots, [\langle\!\langle N_1 \rangle\!\rangle/x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle/x_n][\![x_n : A_n]\!]^- \tag{2}$$

M has the form $\lambda x_1 : A_1 \ldots \lambda x_n : A_n.M'$. Since $M : \Pi x_1 : A_1 \ldots \Pi x_n : A_n.B$ is in Γ the formula below (the translation of this context item using $[\![]\!]^+$) is in $[\![\Gamma]\!]^+$.

$$\forall x_1\,([\![x_1 : A_1]\!]^+ \supset \ldots \forall x_n\,([\![x_n : A_n]\!]^+ \supset (\mathit{hastype}\ \langle\!\langle M' \rangle\!\rangle\ \langle\!\langle B \rangle\!\rangle)) \ldots)$$

By Definition 2 applied to this formula with instances $\langle\!\langle N_1 \rangle\!\rangle, \ldots, \langle\!\langle N_n \rangle\!\rangle$ for the variables $x_1, \ldots, x_n$, we know that the following pair is in $\left|[\![\Gamma]\!]^+\right|_\Sigma$.

$$\Big\langle \{[\langle\!\langle N_1 \rangle\!\rangle / x_1][\![x_1 : A_1]\!]^+, \ldots, [\langle\!\langle N_1 \rangle\!\rangle / x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle / x_n][\![x_n : A_n]\!]^+\},$$
$$(\mathit{hastype}\ [\langle\!\langle N_1 \rangle\!\rangle / x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle / x_n]\langle\!\langle M' \rangle\!\rangle\ [\langle\!\langle N_1 \rangle\!\rangle / x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle / x_n]\langle\!\langle B \rangle\!\rangle)\Big\rangle$$

Hence, by Theorem 3, the fact that the formulas (2) are provable from Σ and $[\![\Gamma]\!]^+$, and the fact that the positive and negative translation are identical on these judgments, we can conclude that the formula on the right of this pair is provable from Σ and $[\![\Gamma]\!]^+$. By Lemmas 4 and 5, the following hold.

$$[\langle\!\langle N_1 \rangle\!\rangle / x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle / x_n]\langle\!\langle M' \rangle\!\rangle =_\beta \langle\!\langle (M N_1 \ldots N_n)^\beta \rangle\!\rangle$$
$$[\langle\!\langle N_1 \rangle\!\rangle / x_1, \ldots, \langle\!\langle N_n \rangle\!\rangle / x_n]\langle\!\langle B \rangle\!\rangle =_\beta \langle\!\langle ([N_1/x_1, \ldots, N_n/x_n]B)^\beta \rangle\!\rangle$$

Thus the formula on the right of the above pair is equivalent to (1) and we have our result. The case when M is a variable, and the case when the last rule in the proof of the LF′ assertion is (APP-FAM) are similar to this case.

The proof of the backward direction is by induction on the structure of the term on the left in α, and is similar to the proof of the forward direction. The proof of the case when the term on the left is an abstraction or Π relies on the fact that there is a sequent proof of the corresponding hh^ω formula of the form described by Theorem 1. The proof of the case when the term on the left is an application uses Theorem 3. ■

The correctness of the translation $[\![]\!]$ is stated as the following corollary of this theorem. We state it with respect to the unmodified canonical LF.

Corollary 8 (Correctness of Translation II)
Let Γ be a canonical context and α a canonical judgment. Then Γ is a valid context and $\Gamma \vdash \alpha$ is provable in LF iff $\Sigma_{LF}; \emptyset \vdash_I [\![\Gamma; \alpha]\!]$ holds.

7 Encoding a Specification of First-Order Logic

In this section, we consider some further examples from an LF specification of natural deduction in first-order logic. We begin by illustrating the translation of context items specifying some of the inference rules. We then consider some example LF judgments provable from this context, and discuss both proof checking and theorem proving of the corresponding goals in hh^ω.

Note that in general, formulas obtained by translating context items have the form on the left below, but can be rewritten to have the form on the right:

$$\forall X_1\ (G_1 \supset \ldots \forall X_n\ (G_n \supset D) \ldots) \qquad \forall X_1 \ldots \forall X_n\ (G_1 \wedge \cdots \wedge G_n \supset D)$$

where $n \geq 0$, $X_1, \ldots, X_n$ are variables, and $G_1, \ldots, G_n, D$ are hh^ω formulas. (Here we assume that for $i = 1, \ldots, n$, $X_{i+1}, \ldots, X_n$ do not appear free in G_i). For readability, we

will write hh^ω formulas in the examples in this section simply as $G_1 \wedge \cdots \wedge G_n \supset D$ (or just D when $n = 0$), and assume implicit universal quantification over all free variables written as capital letters.

The fragment of an LF specification for first-order logic that we are concerned with is the following.

$$
\begin{array}{rl}
i & : \mathsf{Type} \\
\mathit{form} & : \mathsf{Type} \\
\mathit{true} & : \mathit{form} \rightarrow \mathsf{Type} \\
\wedge^* & : \mathit{form} \rightarrow \mathit{form} \rightarrow \mathit{form} \\
\supset^* & : \mathit{form} \rightarrow \mathit{form} \rightarrow \mathit{form} \\
\forall^* & : (i \rightarrow \mathit{form}) \rightarrow \mathit{form} \\
\wedge^*\text{-I} & : \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.(\mathit{true}\ A) \rightarrow (\mathit{true}\ B) \rightarrow (\mathit{true}\ A \wedge^* B) \\
\wedge^*\text{-E}_1 & : \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.(\mathit{true}\ A \wedge^* B) \rightarrow (\mathit{true}\ A) \\
\wedge^*\text{-E}_2 & : \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.(\mathit{true}\ A \wedge^* B) \rightarrow (\mathit{true}\ B) \\
\forall^*\text{-E} & : \Pi A{:}i \rightarrow \mathit{form}.\Pi t{:}i.(\mathit{true}\ \forall^* A) \rightarrow (\mathit{true}\ At) \\
\forall^*\text{-I} & : \Pi A{:}i \rightarrow \mathit{form}.(\Pi y{:}i.(\mathit{true}\ Ay)) \rightarrow (\mathit{true}\ \forall^* A) \\
\supset^*\text{-I} & : \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.((\mathit{true}\ A) \rightarrow (\mathit{true}\ B)) \rightarrow (\mathit{true}\ A \supset^* B) \\
\supset^*\text{-E} & : \Pi A{:}\mathit{form}.\Pi B{:}\mathit{form}.(\mathit{true}\ A \supset^* B) \rightarrow (\mathit{true}\ A) \rightarrow (\mathit{true}\ B)
\end{array}
$$

For readability, we do not always present context items in canonical form. The corresponding canonical term can always be easily deduced. For example, to apply the translation to the inference rules for universal quantification, the term $(\forall^* A)$ must be replaced by $(\forall^* \lambda x{:}i Ax)$.

First, consider the $\forall^*$-elimination rule specified by $\forall^*$-E and its type. Its translation (using $[\![]\!]^+$) is the following formula.

$$
\begin{array}{l}
\forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ Ay\ \mathit{form})) \wedge (\mathit{hastype}\ t\ i) \wedge \\
\quad (\mathit{hastype}\ P\ (\mathit{true}\ \forall^* A)) \supset (\mathit{hastype}\ (\forall^*\text{-E}\ A\ t\ P)\ (\mathit{true}\ At))
\end{array}
$$

This formula reads: if for arbitrary y of type i, Ay is a formula, and if t is a term of type i and P is a proof of $\forall^* A$, then the term $(\forall^*\text{-E}\ A\ t\ P)$ is a proof of the formula At. Note that, as in the translation of the $\forall^*$ connective given in Section 5, A is a function at the meta-level having syntactic type $ob \rightarrow ob$. It maps first-order terms to formulas just as it does at the object-level. We next consider the translation of the $\forall^*$-I rule as the following formula.

$$
\begin{array}{l}
\forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ Ay\ \mathit{form})) \wedge \\
\quad \forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ Py\ (\mathit{true}\ Ay))) \supset \\
\quad\quad (\mathit{hastype}\ (\forall^*\text{-I}\ A\ P)\ (\mathit{true}\ \forall^* A))
\end{array}
$$

This clause provides the following description of the information contained in the dependent type: if for arbitrary y of type i, Ay is a formula and Py is a proof of Ay, then the term $(\forall^*\text{-I}\ A\ P)$ is a proof of $\forall^* A$. Here, both A and P are functions at the meta-level having syntactic type $ob \rightarrow ob$. Again, A maps first-order terms to formulas, while P maps first-order terms to proofs. As a final inference rule example, consider the

declaration for $\supset^*$-I, which translates to the following formula.

$$(\mathit{hastype}\ A\ \mathit{form}) \wedge (\mathit{hastype}\ B\ \mathit{form}) \wedge \\ \forall q((\mathit{hastype}\ q\ (\mathit{true}\ A)) \supset (\mathit{hastype}\ Pq\ (\mathit{true}\ B))) \supset \\ (\mathit{hastype}\ (\supset^*\text{-I}\ A\ B\ P)\ (\mathit{true}\ A \supset^* B))$$

This formula reads: if A and B are formulas and P is a function which maps an arbitrary proof q of A to the proof Pq of B, then the term $(\supset^*\text{-I}\ A\ B\ P)$ is a proof of $A \supset^* B$. Note that P in this formula is a function which maps proofs to proofs.

We consider an example from [21] which is provable in the LF specification for natural deduction. The following LF type represents the fact that in first-order logic, a universal quantifier can be pulled outside a conjunction.

$$\Pi A{:}i \rightarrow \mathit{form}.\Pi B{:}i \rightarrow \mathit{form}. \\ (\mathit{true}\ (\forall^* A\ \wedge^*\ \forall^* B)) \rightarrow (\mathit{true}\ \forall^*(\lambda x{:}i(Ax \wedge^* Bx)))$$

Let the term T be the following LF term of this type, which represents a natural deduction proof of this fact.

$$\lambda A{:}i \rightarrow \mathit{form}.\lambda B{:}i \rightarrow \mathit{form}.\lambda p{:}(\mathit{true}\ (\forall^* A\ \wedge^*\ \forall^* B)). \\ (\forall^*\text{-I}\ \ \lambda x{:}i.(Ax \wedge^* Bx)\ \ \lambda x{:}i.(\wedge^*\text{-I}\ \ Ax\ Bx \\ (\forall^*\text{-E}\ A\ x\ (\wedge^*\text{-E}_1\ \forall^* A\ \ \forall^* B\ \ p))\ (\forall^*\text{-E}\ B\ x\ (\wedge^*\text{-E}_2\ \forall^* A\ \ \forall^* B\ \ p))))$$

Let T' be the following simply typed λ-term of type $(ob \rightarrow ob) \rightarrow (ob \rightarrow ob) \rightarrow ob \rightarrow ob$.

$$\lambda A{:}ob \rightarrow ob.\lambda B{:}ob \rightarrow ob.\lambda p{:}ob. \\ (\forall^*\text{-I}\ \ \lambda x{:}ob.(Ax \wedge^* Bx)\ \ \lambda x{:}ob.(\wedge^*\text{-I}\ \ Ax\ Bx \\ (\forall^*\text{-E}\ A\ x\ (\wedge^*\text{-E}_1\ \forall^* A\ \ \forall^* B\ \ p))\ (\forall^*\text{-E}\ B\ x\ (\wedge^*\text{-E}_2\ \forall^* A\ \ \forall^* B\ \ p))))$$

The encoding of the above judgment using $[\![]\!]^-$ is an hh^ω formula equivalent to the conjunction of the three formulas below, which are provable from the set of formulas encoding the entire LF context specifying natural deduction in first-order logic.

$$(\mathit{istype}\ i) \wedge \forall y((\mathit{hastype}\ y\ i) \supset (\mathit{istype}\ \mathit{form}))$$

$$\forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ Ay\ \mathit{form})) \wedge \\ \forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ By\ \mathit{form})) \supset (\mathit{istype}\ (\mathit{true}\ (\forall^* A\ \wedge^*\ \forall^* B)))$$

$$\forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ Ay\ \mathit{form})) \wedge \\ \forall y((\mathit{hastype}\ y\ i) \supset (\mathit{hastype}\ By\ \mathit{form})) \wedge \\ (\mathit{hastype}\ p\ (\mathit{true}\ (\forall^* A\ \wedge^*\ \forall^* B))) \supset \\ (\mathit{hastype}\ (T'ABp)\ (\mathit{true}\ \forall^*(\lambda x{:}ob.(Ax \wedge^* Bx))))$$

Once a fact is proved it can be considered a part of the context and used to prove new judgments. In this case, the translation of the above judgment as a context item is the latter of the three formulas above. Thus this formula can be added as an assumption and used in proving new hh^ω goals. For example, consider the LF type below.

$$\mathit{true}((\forall^* r\ \wedge^*\ \forall^* s) \supset^* (ra \wedge^* sa))$$

(We assume that a is a constant and r, s are unary predicates in our first-order logic. Thus the context contains $a : i, r : i \rightarrow \mathit{form}, s : i \rightarrow \mathit{form}$, and the set of hh^ω formulas contains their corresponding translations.) The following two LF terms represent proofs of this fact. The first uses the above LF judgment as a lemma.

$$(\supset^*\text{-I}\ (\forall^* r\ \wedge^*\ \forall^* s)\ (ra \wedge^* sa)\ \lambda p{:}(\mathit{true}\ (\forall^* r\ \wedge^*\ \forall^* s)).\\ (\forall^*\text{-E}\ \ \lambda x{:}i.(rx \wedge^* sx)\ \ a\ (Trsp)^\beta)$$

$$(\supset^*\text{-I}\ (\forall^* r\ \wedge^*\ \forall^* s)\ (ra \wedge^* sa)\ \lambda p{:}(\mathit{true}\ (\forall^* r\ \wedge^*\ \forall^* s)).(\wedge^*\text{-I}\ ra\ sa\\ (\forall^*\text{-E}\ r\ a\ (\wedge^*\text{-E}_1\ \forall^* r\ \ \forall^* s\ \ p))\ (\forall^*\text{-E}\ s\ a\ (\wedge^*\text{-E}_2\ \forall^* r\ \ \forall^* s\ \ p))))$$

These judgments translate to the following two provable hh^ω formulas.

$$(\mathit{hastype}\ (\supset^*\text{-I}\ (\forall^* r\ \wedge^*\ \forall^* s)\ (ra \wedge^* sa)\\ \lambda p{:}\mathit{ob}.(\forall^*\text{-E}\ \ \lambda x{:}\mathit{ob}.(rx \wedge^* sx)\ \ a\ (T'rsp))\\ (\mathit{true}\ (\forall^* r\ \wedge^*\ \forall^* s)\ \supset^* (ra \wedge^* sa))$$

$$(\mathit{hastype}\ (\supset^*\text{-I}\ (\forall^* r\ \wedge^*\ \forall^* s)\ (ra \wedge^* sa)\ \lambda p{:}\mathit{ob}.(\wedge^*\text{-I}\ ra\ sa\\ (\forall^*\text{-E}\ r\ a\ (\wedge^*\text{-E}_1\ \forall^* r\ \ \forall^* s\ \ p))\\ (\forall^*\text{-E}\ s\ a\ (\wedge^*\text{-E}_2\ \forall^* r\ \ \forall^* s\ \ p))))\\ (\mathit{true}\ (\forall^* r\ \wedge^*\ \forall^* s)\ \supset^* (ra \wedge^* sa))$$

With respect to the interpreter described in Section 4, we will say that an hh^ω formula with no logic variables is *closed*. The formulas we obtain by applying the translation, for example, are all closed. Proving one of the above two formulas, for instance, corresponds to verifying that the closed term represents a natural deduction proof of the first-order formula in the closed type, *i.e.*, proving closed formulas corresponds to object-level proof checking. The deterministic interpreter described in Section 4 is in fact sufficient to prove such goals. Each BACKCHAIN step will produce new closed subgoals. Consider the first of the two formulas above. In proving this formula, we obtain a subgoal of the form:

$$(\mathit{hastype}\ (T'rsp)\ (\mathit{true}\ (\forall\lambda x{:}\mathit{ob}.(rx \wedge^* sx)))).$$

The term at the head of (the normal form of) $(T'rsp)$ is $\forall^*$-I. At this point in the proof there will be two possible definite clauses that can be used in backchaining: the translation of the $\forall^*$-I context item, and the translation of the lemma T, and either will lead to proof of the subgoal. In fact, for proof checking, we can restrict the set of definite clauses used to those obtained by translating context items that introduce new variables, discarding those that translate context lemmas, and still retain a complete program with respect to a deterministic control. In this restricted setting, at each step depending on the constant at the head of the term, there will be exactly one clause that can be used in backchaining.

To use such a set of hh^ω formulas for object-level theorem proving, we simply use a logic variable in the first argument to the *hastype* predicate. For example, to prove the first-order formula $(\forall^* r\ \wedge^*\ \forall^* s)\ \supset^* (ra \wedge^* sa)$, we begin with the goal:

$$(\mathit{hastype}\ \mathtt{M}\ (\mathit{true}\ (\forall^* r\ \wedge^*\ \forall^* s)\ \supset^* (ra \wedge^* sa)))$$

where M is a logic variable to be instantiated with a term of the given type. A closed instance of M can easily be mapped back to an LF term having the given type. As discussed in Section 2, depth-first search is not sufficient for such a theorem proving goal since there may often be many definite clauses to choose from to use in backchaining. For example, for a subgoal of the form:

$$(\mathit{hastype}\ \mathtt{M}'\ (\mathit{true}\ (\forall\lambda x{:}ob.(rx \wedge^* sx))))$$

among the options available are backchaining on the clause for the lemma T or backchaining directly on the clause for $\forall^*$-I. As discussed earlier, the tactic environment of [7] provides an environment in which such choices can be made.

8 Conclusion

We have not yet considered the possibility of translating hh^ω formulas into LF. This translation is particularly simple. Let Σ be a signature for hh^ω and let $\mathcal{P}$ be a set of Σ-formulas. For each primitive type τ other than o in S, the corresponding LF judgment is τ : Type. For each non-predicate constant c of type τ in Σ, the corresponding LF judgment is $c : \tau$. For each predicate constant p of type $\tau_1 \to \cdots \to \tau_n \to o \in \Sigma$, the corresponding LF judgment is $p : \tau_1 \to \cdots \to \tau_n \to$ Type. Finally, let $D \in \mathcal{P}$ and let k be a new constant not used in the translation to this point. Then the corresponding LF judgment is $k : D'$ where D' is essentially D with $B_1 \supset B_2$ written as $\Pi x : B_1.B_2$ and $\forall_\tau x\ B$ written as $\Pi x{:}\tau.B$.

Notice that the translation presented in this paper works via recursion over the structure of types. Thus, λ-calculi that contain quantification over types such as the polymorphic λ-calculus or the Calculus of Constructions cannot be directly translated in this manner. For example, we cannot define the same notion of base type. Translating A : Type when A is a base type, for instance, results in an atomic formula for the *istype* predicate. In systems with quantification over types, whether or not A is a base type may depend on its instances, and cannot be determined at the time of translation.

The translation we have described provides a method of directly translating an LF specification, so that there is one hh^ω formula corresponding to each LF context item. Since each context item represents a concept of the logic being specified, in the resulting proof checkers and theorem provers, each (BACKCHAIN) step is on a clause for a particular constant representing an object-level notion. Another approach to implementing LF specifications is to implement the inference rules of LF directly as hh^ω formulas, coding the provability relation directly into the meta-language. An LF context specifying a particular logic would serve as a parameter to such a specification. Such an approach adds one level of indirection in implementing object logics since now each (BACKCHAIN) step corresponds to the application of an LF rule. This approach to implementing typed λ-calculi is taken in [8], where it is also shown that it can be applied to systems with quantification over types.

Such an approach requires an encoding of terms at all levels of the calculus being specified. In LF, for instance, meta-level constants for the various notions of application and abstraction must be introduced. For example, at the level of types a constant of type $ty \to ob \to ty$ can be introduced to represent application, while constants of type

$(ob \rightarrow ty) \rightarrow ty$ can be introduced for Π and λ-abstraction. A coding of the convertibility relation on terms is also required in this setting. Note that the above simple types have order 1 and 2 respectively. In fact hh^2 is all that is required to encode provability of typed λ-calculi in this manner. In [5], using such an encoding on terms, it was shown that a direct encoding of LF specifications using the approach in this paper can be defined in just hh^2. The proofs of the correctness of that encoding are similar to those presented here.

In [12], a similar approach based on recursion over types is adopted to implement a subset of the Calculus of Constructions. In the meta-language used there, terms are the terms of the Calculus of Constructions, and a simple language of clauses over these terms is defined. During goal-directed proof, when a new assumption is introduced, the clause corresponding to this assumption is added dynamically and is then available for backchaining. In this way, certain forms of quantification over types can be handled. Such an approach can be implemented in λProlog by implementing the translation as a λProlog program and performing the translation dynamically as types become instantiated to obtain new assumptions which can be used in subsequent proof checking and theorem proving subgoals.

In the Elf programming language [21], a logic programming language is described that gives operational interpretations directly to LF types similar to the way in which the interpreter described in Section 4 gives operational interpretations to the connectives of hh^ω. Logic variables are also used in this implementation, and the more complex operation of unification on LF terms is required. The LF specification for first-order logic discussed in Section 7, for example, can serve directly as a program in this language. The operational behavior, of such a program, although similar to the execution of an hh^ω specification, has several differences. For instance, certain operations which are handled directly at the meta-level by unification on types in an Elf implementation are expressed explicitly as type-checking subgoals in the hh^ω formulas, and thus handled by logic programming search. For example, consider a goal of the form $(true\ A \supset^* B)$ in the first-order logic specification. In Elf, before backchaining on the context item specifying the $\supset^*$-introduction rule, the interpreter verifies that $A \supset^* B$ has type *form*. In the corresponding hh^ω program, the term $A \supset^* B$ in the head of the clause translating the $\supset^*$-I context item will unify with any term of type *ob*. It is the subgoals $(hastype\ A\ form)$ and $(hastype\ B\ form)$ which will succeed or fail depending on whether A and B represent first-order formulas. In addition, when such programs are used as theorem provers, LF proofs are built at the meta-level by Elf, whereas they are explicit arguments to the *hastype* predicate in hh^ω specifications and are built by unification on simply typed λ-terms.

Acknowledgements

The author would like to thank Dale Miller, Frank Pfenning, and Randy Pollack for helpful comments and discussions related to the subject of this paper.

References

[1] Arnon Avron, Furio A. Honsell, and Ian A. Mason. Using typed lambda calculus to implement formal systems on a machine. Technical Report ECS-LFCS-87-31, Laboratory for the Foundations of Computer Science, University of Edinburgh, June 1987.

[2] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[3] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[4] N.G. deBruijn. A survey of the project AUTOMATH. In *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606. Academic Press, New York, 1980.

[5] Amy Felty. *Specifying and Implementing Theorem Provers in a Higher-Order Logic Programming Language.* PhD thesis, University of Pennsylvania, August 1989.

[6] Amy Felty. A logic program for transforming sequent proofs to natural deduction proofs. In Peter Schroeder-Heister, editor, *Proceedings of the 1989 International Workshop on Extensions of Logic Programming*, Tübingen, West Germany. Springer-Verlag LNAI series, 1991.

[7] Amy Felty and Dale Miller. Specifying theorem provers in a higher-order logic programming language. In *Ninth International Conference on Automated Deduction*, Argonne, IL, May 1988.

[8] Amy Felty and Dale Miller. A meta language for type checking and inference: An extended abstract. Presented at the 1989 Workshop on Programming Logic, Bålstad, Sweden, 1989.

[9] Amy Felty and Dale Miller. Encoding a dependent-type λ-calculus in a logic programming language. In *Tenth International Conference on Automated Deduction*, Kaiserslautern, Germany, July 1990.

[10] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Second Annual Symposium on Logic in Computer Science*, Ithaca, NY, June 1987.

[11] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. Technical Report CMU-CS-89-173. To appear, 1989.

[12] Leen Helmink. Resolution and type theory. In *Proceedings of the European Symposium on Programming*, Copenhagen, 1990.

[13] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinatory Logic and Lambda Calculus.* Cambridge University Press, 1986.

[14] William A. Howard. The formulae-as-type notion of construction, 1969. In *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism.* Academic Press, 1980.

[15] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Scicnce*, 1:27–57, 1975.

[16] Per Martin-Löf. *Intuitionistic Type Theory.* Studies in Proof Theory Lecture Notes. BIBLIOPOLIS, Napoli, 1984.

[17] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. To appear in the *Annals of Pure and Applied Logic.*

[18] Gopalan Nadathur and Dale Miller. An overview of λProlog. In K. Bowen and R.. Kowalski, editors, *Fifth International Conference and Symposium on Logic Programming.* MIT Press, 1988.

[19] Gopalan Nadathur and Dale Miller. Higher-order horn clauses. *Journal of the ACM*, 37(4):777 – 814, October 1990.

[20] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, September 1989.

[21] Frank Pfenning. Elf: A language for logic definition and verified metaprogramming. In *Fourth Annual Symposium on Logic in Computer Science*, Monterey, CA, June 1989.

A Full and Canonical LF

In this section, we show the correspondence between canonical LF as presented in Section 3, and full LF as presented in [10]. The rules of full LF are the rules of Figure 1 in Section 3 except for (FAM-LEMMA) and (OBJ-LEMMA) plus the application and conversion rules given in Figure 7 which replace the application rules in Figure 2. The following two

$$\frac{x{:}K \in \Gamma}{\Gamma \vdash x : K}\,(\text{VAR-KIND}) \qquad \frac{x{:}A \in \Gamma}{\Gamma \vdash x : A}\,(\text{VAR-FAM})$$

$$\frac{\Gamma \vdash A : \Pi x{:}B.K \qquad \Gamma \vdash M : B}{\Gamma \vdash AM : [M/x]K}\,(\text{APP-FAM})$$

$$\frac{\Gamma \vdash M : \Pi x{:}A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : [N/x]B}\,(\text{APP-OBJ})$$

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash K' \text{ kind} \qquad K =_\beta K'}{\Gamma \vdash A : K'}\,(\beta\text{-KIND})$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A' \text{ kind} \qquad A =_\beta A'}{\Gamma \vdash M : A'}\,(\beta\text{-FAM})$$

Figure 7: Full LF application and conversion rules

properties of full LF are shown to hold in [10], and will be used in proving the results below.

Lemma 9 (Subderivation)

1. If $\Gamma \vdash A : K$ holds then $\Gamma \vdash K$ kind also holds.
2. If $\Gamma \vdash M : A$ holds then $\Gamma \vdash A :$ Type also holds.

Lemma 10 (Subject Reduction)

1. If $\Gamma \vdash K$ kind holds and K β-reduces to K', then $\Gamma \vdash K'$ kind also holds.
2. If $\Gamma \vdash A : K$ holds and A β-reduces to A', then $\Gamma \vdash A' : K$ also holds.
3. If $\Gamma \vdash M : A$ holds and M β-reduces to M', then $\Gamma \vdash M' : A$ also holds.

We establish some further properties about canonical and full LF that will be needed to show the correspondence between these two systems.

Definition 11 Let Γ be a valid context (in canonical or full LF), and $x{:}P$ an item in Γ. We define the function C which maps variable x and context Γ to a canonical term. P has the form $\Pi x_1 : A_1 \ldots \Pi x_n : A_n.Q$ where $n \geq 0$ and Q is Type or a base type. For $i = 1, \ldots, n$, let Γ_i be the context $\Gamma, x_1{:}A_1, \ldots, x_i{:}A_i$. We define $C(x, \Gamma)$ to be the term:

$$\lambda x_1 : A_1 \ldots \lambda x_n : A_n.x(C(x_1, \Gamma_1)) \ldots (C(x_n, \Gamma_n)).$$

(We will abbreviate $\mathcal{C}(x, \Gamma)$ as $\mathcal{C}(x)$ in the remainder of this section, since Γ can always be determined from context.)

The following lemma holds for both canonical and full LF.

Lemma 12 Let Γ be a valid context containing $x : P$ where P is canonical. Then $\Gamma \vdash \mathcal{C}(x) : P$ is provable.

Proof: The proof is by induction on the structure of P and relies on the fact that for any variable z and well-typed canonical term Q, $([\mathcal{C}(z)/z]Q)^{\beta} = Q$. ■

Using this lemma, the following result about canonical LF can be proven.

Lemma 13 If $\Gamma_1, x : A, P : Q, \Gamma_2$ is a valid context with $P : Q$ a context lemma, and $\Gamma_1, x : A, P : Q, \Gamma_2 \vdash \alpha$ is provable in canonical LF, then $\Gamma_1, \lambda x : A.P : \Pi x : A.Q, x : A, \Gamma_2$ is a valid context and the assertion $\Gamma_1, \lambda x : A.P : \Pi x : A.Q, x : A, \Gamma_2 \vdash \alpha$ is provable.

Proof: We let Γ' and Γ'' be the contexts $\Gamma_1, x : A, P : Q, \Gamma_2$ and $\Gamma_1, \lambda x : A.P : \Pi x : A.Q, x : A, \Gamma_2$, respectively. We first prove the lemma with the added assumption that Γ'' is a valid context by induction on the height of a derivation of $\Gamma' \vdash \alpha$. The only non-trivial case occurs when the context item from Γ' used in an application of (APP-OBJ) or (APP-FAM) is $P : Q$. To show $\Gamma'' \vdash \alpha$, we use the corresponding item $\lambda x : A.P : \Pi x : A.Q$ from Γ'', with the additional hypothesis $\Gamma'' \vdash \mathcal{C}(x) : A$, which we know to be provable by Lemma 12. Using this result, the proof that Γ'' is valid is by a straightforward induction on the length of Γ_2. ■

Definition 14 A *canonical derivation* in full LF is a derivation such that the following hold.

1. The assertion at the root is pre-canonical (*i.e.*, its normal form is canonical).
2. All assertions in the derivation except for those that occur as the conclusion of (VAR-KIND) or (VAR-OBJ), or as the conclusion *and* left premise of (APP-FAM) or (APP-OBJ) are pre-canonical.
3. In all assertions that occur as the conclusion, but not a left premise of (APP-FAM) or (APP-OBJ), the term on the right of the judgment is Type or a base type.

We will only consider canonical derivations in full LF when demonstrating the relative soundness and completeness of canonical LF. By imposing this restriction we are eliminating exactly those derivations such that a term used on the left of an application is not applied to the maximum number of arguments. A derivation that does not meet this requirement can, in fact, be mapped in a straightforward manner to one that does by introducing context items of the appropriate types and discharging them with an (ABS) rule. For example, a derivation of $\Gamma \vdash P : \Pi x : A.Q$ where P is not an abstraction can be mapped to a derivation of $\Gamma, x : A \vdash P : \Pi x : A.Q$ to which we can apply the

corresponding (APP) rule to obtain $\Gamma, x:A \vdash P(C(x)) : Q$, followed by an (ABS) rule to obtain $\Gamma \vdash \lambda x:A.P(C(x)) : \Pi x:A.Q$.

We now define by induction an operation $\mathcal{L}$ which maps a derivation in full LF to a sequence of typing judgments. As we will see, the sequence of judgments associated to a canonical derivation is exactly the set of lemmas that will be added to the context to obtain the corresponding derivation in canonical LF.

Definition 15 $\mathcal{L}$ maps a derivation in full LF to a sequence of typing judgments Δ defined by induction on the derivation as follows.

- If the last rule in the derivation is (TYPE-KIND), (VAR-KIND), (VAR-FAM), or (EMPTY-CTX), then Δ is the empty sequence.
- If the last rule is a (PI) or (ABS) rule, then let Δ_1 be the sequence associated by $\mathcal{L}$ to the derivation of the left premise $\Gamma \vdash A : \mathsf{Type}$, and Δ_2 be the sequence associated to the derivation of the right premise. Let Δ_2' be the sequence that replaces every judgment $P:Q$ in Δ_2 with $\lambda x:A^\beta.P : \Pi x:A^\beta.Q$. Then Δ is Δ_1, Δ_2'.
- If the last rule is an (APP) rule, then let Δ_1 be the sequence associated by $\mathcal{L}$ to the derivation of the left premise $\Gamma \vdash P : Q$, and Δ_2 be the sequence associated to the derivation of the right premise. If the left premise is pre-canonical, and is not the conclusion of another (APP) rule or of a (VAR) rule, then Δ is $\Delta_1, P^\beta : Q^\beta, \Delta_2$. Otherwise, Δ is Δ_1, Δ_2.
- If the last rule is a β rule, Δ is the sequence associated to the derivation of the leftmost premise.
- If the last rule is (FAM-INTRO) or (OBJ-INTRO), then let Δ_1 and Δ_2 be the sequences associated by $\mathcal{L}$ to the derivation of the left and right premises, respectively. Then Δ is Δ_1, Δ_2.

It can be shown by a straightforward induction on a derivation of $\Gamma \vdash \alpha$ that for each judgment $P:Q$ in the sequence associated to this assertion by $\mathcal{L}$, $\Gamma \vdash P : Q$ holds.

Let $x_1 : P_1, \ldots, x_n : P_n$ be a valid context in full LF. For $i = 1, \ldots, n$, we denote the subcontext whose last element is $x_i : P_i$ as Γ_i. Given a derivation of $\vdash \Gamma_n$ context, for $i = 1, \ldots, n$, let Δ_i be the context associated to the subderivation of $\Gamma_{i-1} \vdash P_i$ kind or $\Gamma_{i-1} \vdash P_i : \mathsf{Type}$. We say that the context $\Delta_1, x_1 : P_1^\beta, \ldots, \Delta_n, x_n : P_n^\beta$ is the *extended normal context* associated to this derivation.

Theorem 16 (Completeness of Canonical LF) Let Γ be a context and α a judgment such that $\vdash \Gamma$ context and $\Gamma \vdash \alpha$ have canonical derivations in full LF. Let Γ' be the extended normal context associated to the derivation of $\vdash \Gamma$ context, and let Δ be the set of typing judgments associated to the derivation of $\Gamma \vdash \alpha$ by the function $\mathcal{L}$. Then Γ', Δ is a valid context and $\Gamma', \Delta \vdash \alpha^\beta$ is provable in canonical LF.

Proof: We first prove the above statement under the additional hypotheses that Γ' is a valid context in canonical LF. Using this result, it can be proved by a straightforward

induction on the length of Γ that Γ' is valid. The proof is by induction on the height of a canonical derivation in full LF of $\Gamma \vdash \alpha$. For the case when the last rule is an (APP) rule, we must consider the subproof that contains a series of n (APP) rules, where $n \geq 1$ and the leftmost premise $\Gamma \vdash P : Q$ is not the conclusion of an (APP) rule. We consider the case when this premise is not the conclusion of a (VAR) rule. Thus P and Q are pre-canonical. First, we show that Γ', Δ is a valid context. Let Δ_0 be the sequence associated to $\Gamma \vdash P : Q$ by $\mathcal{L}$, and for $i = 1, \ldots, n$, let Δ_i be the sequence associated with the right premise in the ith (APP) rule application. Then Δ is $\Delta_0, P^\beta : Q^\beta, \Delta_1, \ldots, \Delta_n$. By the induction hypothesis applied to $\Gamma \vdash P : Q$, the context Γ', Δ_0 is valid and $\Gamma', \Delta_0 \vdash P^\beta : Q^\beta$ holds. Hence $\Gamma', \Delta_0, P^\beta : Q^\beta$ is a valid context. Also, by the induction hypothesis, for $i = 1, \ldots, n$, Γ', Δ_i is a valid context. Thus, we can conclude that Γ', Δ is valid. We now show $\Gamma', \Delta \vdash \alpha^\beta$ holds. By the induction hypothesis, for $i = 1, \ldots, n$, for each right premise $\Gamma \vdash \alpha_i$ in the series of (APP) rules, $\Gamma', \Delta_i \vdash \alpha_i^\beta$ holds. Clearly $\Gamma', \Delta \vdash \alpha_i^\beta$ also holds. Using the context item $P^\beta : Q^\beta$, we can simply apply the canonical LF rule (APP-FAM) or (APP-OBJ) to these n assertions to obtain that $\Gamma', \Delta \vdash \alpha^\beta$ holds. The case when $\Gamma \vdash P : Q$ is the conclusion of a (VAR) rule is similar.

For the case when the last rule is a (PI) or (ABS) rule, let Δ_1 be the sequence of judgments associated to the left premise $\Gamma \vdash A : \mathsf{Type}$ by $\mathcal{L}$. Let Δ_2 be the sequence associated to the right premise $\Gamma, x : A \vdash \alpha_0$, and Δ_2' be the sequence that contains $\lambda x : A^\beta . P : \Pi x : A^\beta . Q$ for every $P : Q$ in Δ_2. Then Δ is Δ_1, Δ_2'. We first show that $\Gamma', \Delta_1, \Delta_2'$ is a valid context. By the induction hypothesis for the left premise, Γ', Δ_1 is a valid context, and $\Gamma', \Delta_1 \vdash A^\beta : \mathsf{Type}$ holds. Thus, $\Gamma', \Delta_1, x : A^\beta$, the extended normal context associated to $\Gamma, x : A$ is also valid. We now apply the induction hypothesis to the right premise to obtain that $\Gamma', \Delta_1, x : A^\beta, \Delta_2$ is a valid context. By Lemma 13, $\Gamma', \Delta_1, \Delta_2', x : A^\beta$ is also a valid context. Since Γ', Δ is a subcontext of this context, it is valid also. Next, we show that $\Gamma', \Delta_1, \Delta_2' \vdash \alpha^\beta$ holds. By the induction hypothesis for the right premise and Lemma 13, $\Gamma', \Delta_1, \Delta_2', x : A^\beta \vdash \alpha_0^\beta$ holds. Since $\Gamma', \Delta_1 \vdash A^\beta : \mathsf{Type}$ holds, clearly also $\Gamma', \Delta_1, \Delta_2' \vdash A^\beta : \mathsf{Type}$ holds. Thus, by an application of the corresponding canonical LF (PI) or (ABS) rule, $\Gamma', \Delta_1, \Delta_2' \vdash \alpha^\beta$ holds. The remaining cases follow directly from the induction hypothesis and the definition of $\mathcal{L}$. ■

Theorem 17 (Soundness of Canonical LF) Let $\Gamma \vdash \alpha$ be a provable assertion in canonical LF. Let Γ' be the subcontext of Γ containing only variables associated with their types or kinds. Then $\Gamma' \vdash \alpha$ is provable in full LF.

Proof: We first prove the above statement under the additional hypotheses that Γ' is a valid context in full LF and $\Gamma' \vdash P : Q$ for every item $P : Q$ in Γ. Using this result, it can then be shown by induction on the length of Γ that this hypothesis follows from the fact that Γ is valid in canonical LF. We proceed by induction on the height of a canonical LF derivation of $\Gamma \vdash \alpha$. For the (APP) rules, we replace a single application of the rule by a series of n applications of the corresponding rule in full LF. (If n is 0, we simply apply the corresponding (VAR) rule.) If the rule uses a context lemma, we replace it with a full LF derivation of this lemma, which we know to be provable by assumption. The conclusion of each (APP) rule application is a judgment of the form $\Gamma \vdash P : [N/x]Q$ where P and $[N/x]Q$ are not necessarily in β-normal form. We must show that the

corresponding β-normal assertion is also provable. $[N/x]Q$ must be a type or kind by Proposition 9. By Proposition 10, we obtain that $([N/x]Q)^\beta$ is also a type or kind. Then, by the corresponding β rule, $\Gamma \vdash P : ([N/x]Q)^\beta$ is provable. By Proposition 10, we know that $\Gamma \vdash P^\beta : ([N/x]Q)^\beta$ is also provable. The remaining cases follow directly from the induction hypothesis. ■

B LF with Simplified Abstraction Rules

In this section we show that for LF assertions such that all types bound by outermost abstraction in the term on the left in the judgment are valid, the canonical LF system presented in Section 3 is equivalent to LF′, the system obtained by replacing the (ABS-FAM) and (ABS-OBJ) rules with the (ABS-FAM′) and (ABS-OBJ′) rules of Section 6, which drop the left premise $\Gamma \vdash A : \mathsf{Type}$. In such derivations, this premise is redundant.

To prove this result, we need the following transitivity lemma for LF′.

Lemma 18 (Transitivity) If $\Gamma, x : A, \Gamma' \vdash \alpha$ and $\Gamma \vdash N : A$ are provable, then $\Gamma, ([N/x]\Gamma')^\beta \vdash ([N/x]\alpha)^\beta$ is provable.

Proof: The proof is by induction on the structure of proofs. A similar result is stated for the more general presentation of LF in [10]. ■

The following lemma shows that the left premise is redundant in all derivations of assertions such that the term on the left in the judgment is not an abstraction.

Lemma 19 Let Γ be a valid context and $\Gamma \vdash \alpha$ a provable assertion in LF′ that has a proof whose last rule is an application of (APP-FAM) or (APP-OBJ), and that has an application of (ABS-OBJ′) above the root such that there are no other applications of (APP-FAM) or (APP-OBJ) below it. Let Γ' be the context, and $x{:}A$ be the variable and its type bound by λ in the conclusion of this application of (ABS-OBJ′). Then $\Gamma' \vdash A : \mathsf{Type}$ is provable.

Proof: Let $Q : \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.P$ be the context item used in the rule application at the root, and $N_1, \ldots, N_n$ the terms on the right of the colon in the remaining premises. Since there is an (ABS-OBJ′) application above the root, then for some i such that $1 \leq i \leq n$, A_i has the form $\Pi z : B.C$, the corresponding premise of the (APP) rule has the form

$$\Gamma \vdash N_i : ([N_1/x_1, \ldots, N_{i-1}/x_{i-1}]\Pi z{:}B.C)^\beta,$$

where N_i has the form $\lambda z : ([N_1/x_1, \ldots, N_{i-1}/x_{i-1}]B)^\beta.M$, and the rule application at the root of this premise is (ABS-OBJ′). We show that $([N_1/x_1, \ldots, N_{i-1}/x_{i-1}]B)^\beta$ is a type. We know that $\Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.P$ is a type or kind, since Γ is a valid context. A proof of this fact contains a subproof of

$$\Gamma, x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1} \vdash \Pi z{:}B.C : \mathsf{Type}.$$

By Lemma 18, using the premises of the (APP) rule, we can conclude that the assertion $\Gamma \vdash ([N_1/x_1, \ldots, N_{i-1}/x_{i-1}]\Pi z{:}B.C)^\beta : \mathsf{Type}$ is provable. A proof of this fact contains a proof of the desired result. By similar reasoning, all other $x : A$ bound by λ in an application of (ABS-OBJ′) can be shown to be types with respect to the corresponding context. ■

Theorem 20 Let Γ be a context that is valid in LF and LF′, and let α be a judgment.

1. If $\Gamma \vdash \alpha$ is provable in LF, then it is also provable in LF$'$.

2. If $\Gamma \vdash \alpha$ is provable in LF$'$ and all types bound by λ in outermost abstractions in the term on the left in α are valid in LF$'$, then $\Gamma \vdash \alpha$ is provable in LF.

Proof: (1) is proved by a straightforward induction on the height of an LF derivation of $\Gamma \vdash \alpha$. (2) is proved by induction on the structure of the term on the left in α. For the case when the term is an application, the last rule in a derivation must be an (APP) rule. Lemma 19 is required to show that all of the types bound by λ in outermost abstractions in the premises are valid in LF$'$, so that the induction hypothesis can be applied to these assertions. All other cases follow directly from the induction hypothesis. ■

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249 - 6399